

# DL-PA and DCL-PC: model checking and satisfiability problem are indeed in PSPACE

Philippe Balbiani<sup>\*</sup>  
 Andreas Herzig<sup>†</sup>  
 François Schwarzentruher<sup>‡</sup>  
 Nicolas Troquard<sup>§</sup>

## Abstract

We prove that the model checking and the satisfiability problem of both Dynamic Logic of Propositional Assignments DL-PA and Coalition Logic of Propositional Control and Delegation DCL-PC are in PSPACE. We explain why the proof of EXPTIME-hardness of the model checking problem of DL-PA presented in [1, Thm 4] is false. We also explain why the proof of membership in PSPACE of the model checking problem of DCL-PC given in [9, Thm. 4] is wrong.

**Keywords:** Dynamic Logic of Propositional Assignments. Coalition Logic of Propositional Control and Delegation. Model checking. Satisfiability. PSPACE.

## 1 Introduction

Balbani *et al* [1] study a variant of PDL called Dynamic Logic of Propositional Assignments (DL-PA). The latter was introduced in [4] and is a fragment of Tiomkin and Makowsky's extension of PDL by assignments [8]. It is said to be well-behaved because unlike PDL, it is compact, has the interpolation property, and the Kleene star can be eliminated. The logic was partly inspired by the logic of delegation and propositional control DCL-PC presented in [9]. In [1], polynomial translations from DCL-PC to DL-PA and back are proposed.

Between the papers [9], [4] and [1], there have been conflicting results about the complexity of decision problems for DL-PA and DCL-PC, satisfiability checking and model checking. There have also been inadequate proofs for true the-

---

<sup>\*</sup>Université de Toulouse, CNRS, IRIT, F-31062 Toulouse, France

<sup>†</sup>Université de Toulouse, CNRS, IRIT, F-31062 Toulouse, France

<sup>‡</sup>ENS Rennes, Campus de Ker Lann, 35170 BRUZ, France

<sup>§</sup>ISTC-CNR, Trento, Italy. LACL, Université Paris-Est Créteil, France.

orem statements, and there have been wrong proofs for wrong theorem statements. The aim of the present paper is to set the record straight. Specifically:<sup>1</sup>

- The proof in [9] that DCL-PC model checking is in PSPACE is inadequate. It only proves that it is in EXPTIME. A consequence is also that the proof that DCL-PC satisfiability checking is in PSPACE is inadequate, too.
- Following the same proof strategy, the proof in [4] that DL-PA model checking is in PSPACE is inadequate. It only proves that it is in EXPTIME.<sup>2</sup>
- The proof in [1] that DL-PA model checking is EXPTIME-hard is wrong: more precisely, the statement of [1, Thm 4] is wrong.
- The model checking problem and the satisfiability checking problem of DL-PA are both PSPACE-complete.
- The model checking problem and the satisfiability checking problem of DCL-PC are both PSPACE-complete.

## 2 Two dynamic logics

We present DL-PA and DCL-PC which are two interconnected dynamic logics.

### 2.1 Dynamic logic of propositional assignments DL-PA

**Syntax** Let  $PV$  be a countable set of propositional variables (with typical members noted  $p, q$ , etc). The set  $\text{Pgm}(PV)$  of all programs (with typical members noted  $\alpha, \beta$ , etc) and the set  $\text{Fml}(PV)$  of all formulas (with typical members noted  $\phi, \psi$ , etc) are inductively defined as follows:

$$\begin{array}{lcl} \alpha ::= & +p & | \quad -p \quad | \quad (\alpha; \alpha) \quad | \quad (\alpha \cup \alpha) \quad | \quad \alpha^* \quad | \quad \phi? \\ \phi ::= & p & | \quad \perp \quad | \quad [\alpha]\phi \end{array}$$

We define the other Boolean constructs as usual:  $\neg\phi = [\phi?]\perp$ ,  $(\phi \rightarrow \psi) = [\phi?]\psi$ , etc. The formula  $\langle\alpha\rangle\phi$  is obtained as an abbreviation:  $\langle\alpha\rangle\phi = \neg[\alpha]\neg\phi$ . We write  $\alpha^d$  for the sequence of  $\alpha$  repeated  $d$  times. We adopt the standard rules for omission of the parentheses. Let us consider an enumeration  $p_1, p_2, \dots$  of  $PV$ . Program “ $+p$ ” makes proposition  $p$  true and program “ $-p$ ” makes proposition  $p$  false. The number of symbol occurrences in program  $\alpha$  and formula  $\phi$  are respectively noted  $\text{len}(\alpha)$  and  $\text{len}(\phi)$ .

<sup>1</sup> We recall that  $\text{PSPACE} = \text{NPSpace}$  ([5], [6]) and  $\text{APSPACE} = \text{EXPTIME}$  ([2]). In this paper we assume that  $\text{PSPACE}$  is different from  $\text{EXPTIME}$ . If they are equal the whole discussion ends up being a non-issue.

<sup>2</sup> The error is in the published version and is signaled on the website of the conference <http://ijcai.org/papers11>.

**Semantics** A valuation is a subset of  $PV$ , with typical elements  $U, V$ , etc. We inductively define the value of a program  $\alpha$ , in symbols  $\|\alpha\|$ , and the value of a formula  $\phi$ , in symbols  $\|\phi\|$ , as follows:

$$\begin{aligned}
\| + p \| &= \{(U, V) : V = U \cup \{p\}\} \\
\| - p \| &= \{(U, V) : V = U \setminus \{p\}\} \\
\| \alpha; \beta \| &= \{(U, V) : \text{there exists } W \subseteq PV \text{ such that} \\
&\quad (U, W) \in \|\alpha\| \text{ and } (W, V) \in \|\beta\|\} \\
\| \alpha \cup \beta \| &= \|\alpha\| \cup \|\beta\| \\
\| \alpha^* \| &= \{(U, V) : \text{there exist } n \in \mathbb{N} \text{ and } W_0, \dots, W_n \subseteq PV \text{ such that} \\
&\quad U = W_0, (W_0, W_1) \in \|\alpha\|, \dots, (W_{n-1}, W_n) \in \|\alpha\| \text{ and } W_n = V\} \\
\| \phi? \| &= \{(U, V) : U = V \text{ and } V \in \|\phi\|\} \\
\| p \| &= \{U : p \in U\} \\
\| \perp \| &= \emptyset \\
\| [\alpha] \phi \| &= \{U : \text{for all } V \subseteq PV, \text{ if } (U, V) \in \|\alpha\|, \text{ then } V \in \|\phi\|\}
\end{aligned}$$

It follows that  $\| \langle \alpha \rangle \phi \| = \{U : \text{there exists } V \subseteq PV \text{ such that } (U, V) \in \|\alpha\| \text{ and } V \in \|\phi\|\}$ .

## 2.2 Coalition logic of propositional control and delegation DCL-PC

Coalition Logic of Propositional Control and Delegation (DCL-PC) is a logic of *agency*. Let  $PV$  be a countable set of propositional variables and  $\mathbb{A}$  be a finite set of *agents*.

The models of DCL-PC—models of propositional control—are couples  $(V, \xi)$  where  $V$  is a subset of  $PV$  and  $\xi$  maps each propositional variable to one agent in  $\mathbb{A}$ . The function  $\xi$  is a control function. Intuitively, for each proposition  $p$ , the object  $\xi(p)$  denotes the one and only one agent controlling it. Saying that the agent  $\xi(p)$  controls  $p$ , we mean that  $\xi(p)$  can set  $p$  to true and can set  $p$  to false.

The language of DCL-PC extends propositional logic with two families of modalities. One type of modalities is reminiscent of dynamic logics, and thus we have a two-sorted language. In the following grammar,  $i, j \in \mathbb{A}$ , and  $p \in PV$ .

$$\begin{array}{lcl}
\pi & ::= & i \rightsquigarrow_p j \mid (\pi; \pi) \mid (\pi \cup \pi) \mid \pi^* \mid \phi? \\
\phi & ::= & p \mid \perp \mid \Diamond_i \phi \mid \langle \pi \rangle \phi
\end{array}$$

We adopt the standard abbreviations.

To differentiate the truth values of DCL-PC from those of DL-PA, we will denote the value of DCL-PC programs and DCL-PC formulas by  $\|\cdot\|^\#$ .

Atomic delegation programs are of the form  $i \rightsquigarrow_p j$  and are read “ $i$  transfers his control over  $p$  to  $j$ ”. The intuition is that  $i \rightsquigarrow_p j$  is applicable when  $i$  controls

$p$  and that it changes the control function  $\xi$  such that  $j$  gets control over  $p$  (and  $i$  loses it, control being exclusive). Complex delegation programs are defined by means of the standard PDL operators. The interpretation of a delegation program is a binary relation on the set of models of propositional control over  $PV$  and  $\mathbb{A}$ . For atomic programs we have:

$$\|i \rightsquigarrow_p j\|^\# = \{ ((V, \xi), (V, \xi')) : \xi(p) = i, \xi'(p) = j, \text{ and } \xi(q) = \xi'(q) \text{ for } q \neq p \}$$

The interpretation of complex programs is as usual.

The interpretation of DCL-PC formulas is a subset of models of propositional control over  $PV$  and  $\mathbb{A}$ .

$$\|p\|^\# = \{ (V, \xi) : p \in V \}$$

The interpretation of  $\langle \pi \rangle \phi$  is:

$$\|\langle \pi \rangle \phi\|^\# = \{ (V, \xi) : \text{there is } (U, \xi') \text{ such that } ((V, \xi), (U, \xi')) \in \|\pi\|^\# \text{ and } (U, \xi') \in \|\phi\|^\# \}$$

The modality  $\Diamond_i$  allows one to talk about what an agent  $i$  is able to do by changing the truth value of the propositional variables under its control.

$$\|\Diamond_i \phi\|^\# = \{ (V, \xi) : \text{there is } U \text{ such that } (U, \xi) \in \|\phi\|^\# \text{ and for every } p, \text{ if } \xi(p) \neq i \text{ then } p \in V \text{ iff } p \in U \}$$

The interpretation of complex formulas is as usual.

## 2.3 Connection

As announced the two dynamic logics reviewed here are interconnected. In particular, we can apply the algorithms for the decision problems of DL-PA to solve the the decision problems of DCL-PC. Of concern here are four decision problems:

- DL-PA-model checking (*MC*):  
**input:** a valuation  $U$ , and a formula  $\phi \in \mathbf{Fml}(PV)$ ,  
**output:** *yes* if  $U \in \|\phi\|$ , *no* otherwise.
- DL-PA-satisfiability (*SAT*):  
**input:** a formula  $\phi \in \mathbf{Fml}(PV)$ ,  
**output:** *yes* if  $\|\phi\| \neq \emptyset$ , *no* otherwise.
- DCL-PC-model checking  
**input:** a model of propositional control  $(U, \xi)$ , and a DCL-PC formula,

**output:** *yes* if  $(U, \xi) \in \|\phi\|^\#$ , *no* otherwise.

- DCL-PC-satisfiability:

**input:** a DCL-PC formula  $\phi$ ,

**output:** *yes* if  $\|\phi\|^\# \neq \emptyset$ , *no* otherwise.

**Theorem 1** ([1, Section VIII]). *There is a polynomial reduction of DCL-PC-model checking into DL-PA-model checking. There is a polynomial reduction of DCL-PC-satisfiability into DL-PA-satisfiability.*

Hence, the complexity upper bound for a problem of DL-PA will transfer polynomially to a complexity upper bound for the corresponding problem of DCL-PC.

### 3 Issue in the proof of [1, Thm 4]

Theorem 4 in [1] wrongly states that *MC* and *SAT* are EXPTIME-hard. The source of the problem lies in [1, Lemma 1] which wrongly states that *MC* is EXPTIME-hard, proposing an inadequate argument for establishing the existence of a logarithmic-space reduction of the problem PEEK- $G_5$  [7] into *MC*. The claim about *SAT* then comes from an actual logarithmic-space reduction of the problem *MC* into *SAT*.

This section concentrates on the issue with the reduction of the problem PEEK- $G_5$  into *MC*.

An instance of *Peek* is a tuple  $PE = (X_E, X_A, \Phi, V_0, \tau)$  where  $X_E$  and  $X_A$  are finite sets of propositional variables such that  $X_E \cap X_A = \emptyset$ , the idea being that Player *E* controls the variables in  $X_E$  and Player *A* controls the variables in  $X_A$ ;  $\Phi$  is a propositional formula over  $X_E \cup X_A$ ;  $V_0 \subseteq X_E \cup X_A$  indicates which variables are currently true;  $\tau$  is either *A* or *E*, indicating which player makes the next move.

Informally, each instance  $PE = (X_E, X_A, \Phi, V_0, \tau)$  of *Peek* is played as follows. Agents' turns strictly alternate. At their respective turn, Player *E* (resp. *A*) *moves* by changing the truth value of at most one variable of  $X_E$  (resp.  $X_A$ ) in the current valuation, either adding or withdrawing it from the valuation. The game ends when  $\Phi$  first becomes true, in which case we say that Player *E* wins. We say that *Player E has a winning strategy in PE* if she can make a sequence of moves at her turns that ensures to eventually win whatever the moves made by Player *A* at his turn.

The decision problem PEEK- $G_5$  takes as input an instance  $PE = (X_E, X_A, \Phi, V_0, \tau)$  of *Peek*; It outputs *yes*, when Player *E* has a winning strategy in *PE* and *no* otherwise. PEEK- $G_5$  is EXPTIME-complete [7].

In [1, Lemma 1], it was stated that the problem PEEK- $G_5$  on the instance  $PE = (X_E, X_A, \Phi, V_0, \tau)$  returns *no* if and only if *MC* return *yes* on the instance

$(V_{PE}, \varphi_{PE})$ , where:

$$\begin{aligned}
V_{PE} &\stackrel{\text{def}}{=} \begin{cases} V_0 \cup \{\text{nowin}\} & , \text{ when } \tau = A \\ V_0 \cup \{\text{nowin}, \text{elo}\} & , \text{ when } \tau = E \end{cases} \\
\text{moveE} &\stackrel{\text{def}}{=} \text{elo?}; \bigcup_{x \in X_E} (-x \cup +x); -\text{elo} \\
\text{moveA} &\stackrel{\text{def}}{=} -\text{elo?}; \bigcup_{y \in X_A} (-y \cup +y); +\text{elo} \\
\text{move} &\stackrel{\text{def}}{=} (\text{moveE} \cup \text{moveA}); ((\Phi?; -\text{nowin}) \cup \neg\Phi?) \\
\varphi_{PE} &\stackrel{\text{def}}{=} [\text{move}^*](\text{nowin} \rightarrow (\neg\Phi \wedge (\text{elo} \rightarrow [\text{move}]\text{nowin}) \wedge (\neg\text{elo} \rightarrow \langle \text{move} \rangle \text{nowin})))
\end{aligned}$$

This is incorrect. For the anecdote, the mistake was found when one of us figured that if the reduction were actually working, a similar reduction could be done from PEEK- $G_5$  into the problem of model checking CTL formulas over NuSMV models, which is known to be in PSPACE. The implementation of it and the checking of a simple instance indicated the mistake.<sup>3</sup> The instance of Peek considered was  $PE = (X_E, X_A, \Phi, V_0, \tau)$ , where  $X_E = \{p\}$ ,  $X_A = \{q, r\}$ ,  $\Phi = p \wedge q$ ,  $V_0 = \emptyset$  and  $\tau = A$ . Clearly, if  $A$  never adds  $q$  to the valuation  $V_0$ , then  $\Phi$  cannot ever be true. Since  $\tau = A$ , this means that  $E$  has no winning strategy in the game, and PEEK- $G_5$  returns *no* on this instance. However, the problem *MC* also returns *no* on the instance  $(V_{PE}, \varphi_{PE})$ , establishing a counter-example to [1, Lemma 1].

Without this lemma, Proposition 14 in [1] stating that *MC* is EXPTIME-hard has no basis. In turn, Proposition 15 about *SAT* being EXPTIME-hard has no basis either. Theorem 4 in [1] is wrong if  $\text{PSPACE} \neq \text{EXPTIME}$ .

## 4 On the proof of [9, Thm. 4] for PSPACE membership of DCL-PC model checking

In [9], the authors state that the model checking problem for DCL-PC (w.r.t. direct models) is PSPACE-complete. As we shall see later, the result is true in virtue of the algorithm for solving model checking problem for DL-PA (Section 5) and Theorem 1. Nevertheless, the algorithm proposed is alternating, not non-deterministic as claimed in the article. It therefore only allows one to conclude that the DCL-PC model checking problem is in APSPACE and not in PSPACE. This was already pointed out in [1]; we provide a more complete explanation now.

Let us explain why the algorithm is alternating and not non-deterministic. In fact their algorithm is of the following form. Algorithm ‘DCL-PCeval’ of Figure 8, line 5 in [9] negates the Boolean result in the following way:

---

<sup>3</sup>The NuSMV file can be found at this URL <http://www.loa.istc.cnr.it/personal/troquard/SOFTWARES/error-peekdlpa.smv> and its listing is presented in the appendix.

```

function DCL-PCeval( $\phi, \mathcal{M}$ )
  if ... then
     $\vdots$ 
  else if  $\phi = \neg\psi$  then
    return not DCL-PCeval( $\psi, \mathcal{M}$ )
  else
     $\vdots$  (with a call to program-eval)
  endIf
endFunction

```

where ‘program-eval’ (see Fig. 7 in [9]) explicitly mentions non-deterministic choices. But negation implicitly dualizes the algorithm: it transforms true, false, non-deterministic choice, and universal choices into false, true, universal choices, and non-deterministic choice respectively. So the algorithm is in fact alternating.<sup>4</sup>

## 5 A deterministic procedure for DL-PA-model checking and satisfiability problem

Our goal in this paper is to prove the following result.

**Proposition 1.** *The DL-PA-model checking and satisfiability problem are in PSPACE.*

Proposition 1 will be obtained as a direct consequence of Proposition 2 and Claims 1 and 2. As to *SAT*, one can check satisfiability of a formula  $\phi$  by an algorithm which first guesses a valuation  $v$  and then model-checks whether  $v \models \phi$ . This algorithm works in nondeterministic polynomial space NPSPACE, and NPSPACE = PSPACE due to Savitch’s Theorem.

Furthermore, by Theorem 1 we have:

**Corollary 1.** *The DCL-PC-model checking and satisfiability problem are in PSPACE.*

### 5.1 Divide and conquer

Divide and conquer is a familiar algorithmic design technique: for solving a problem, we cut it in several pieces, solve subproblems and combine their results. In the model checking problem for DL-PA, the subproblem to which we will apply divide and conquer is the following one:

**input:** two valuations  $U, V$ , a program  $\alpha$ ;

---

<sup>4</sup> Using the ‘return’ instruction to return the Boolean result of a function is perfectly correct in a *deterministic* algorithm. Nevertheless, when one writes non-deterministic algorithms one should *explicitly* use the ‘reject’ and ‘accept’ instructions that respectively correspond to the rejection and the acceptance state in a Turing machine. Negations are strictly forbidden in a non-deterministic algorithm.

**output:** *yes* if  $(U, V) \in \|\alpha\|$ , no otherwise.

This problem becomes tricky when  $\alpha$  is of the form  $\beta^*$ . As we are concerned by a finite set of propositional variables, let say  $k$  propositional variables, the cardinal of the set of valuations is  $2^k$ . Therefore,  $(U, V) \in \|\beta^*\|$  is equivalent to  $(U, V) \in \|\beta^N\|$  for  $N \in \{0, \dots, 2^k - 1\}$ . In particular, if  $N$  is even,  $(U, V) \in \|\beta^N\|$  iff there exists  $W$  such that  $(U, W) \in \|\beta^{\frac{N}{2}}\|$  and  $(W, V) \in \|\beta^{\frac{N}{2}}\|$ . Thanks to divide and conquer, we are able to design an algorithm that works in polynomial space for the model checking problem in DL-PA.

Actually, the divide and conquer paradigm already appears in the proof of Savitch's theorem ([5], [6]). It has also been recently applied to prove the membership in PSPACE of the model checking of an epistemic formula dealing with agent cameras [3].

## 5.2 Description of the algorithm

Let us assume that the language only contains  $k$  propositional variables. In the sequel, sequences of bits are sequence of length  $k$  whereas “+1” means “+1 modulo  $2^k$ ”. Such sequences will be used to represent valuations. More precisely, the valuation represented by a sequence  $val$  of  $k$  bits makes propositional variable  $p_i$  true iff the  $i$ -th bit of  $val$  is 1. Sequences of  $k$  bits will also be used to represent integers in  $\{0, \dots, 2^k - 1\}$ . In this case, they will be noted by  $d$ ,  $e$ , etc. In the sequel, for all sequences  $d, e$  of  $k$  bits,  $d < e$  will mean that the integer represented by the sequence  $d$  is strictly smaller than the integer represented by the sequence  $e$ . We define the deterministic Boolean function  $REL$  taking as input a bit  $b$ , two valuations  $val$  and  $val'$  and a program  $\alpha$ , the deterministic Boolean function  $MOD$  taking as input a bit  $b$ , a valuation  $val$  and a formula  $\varphi$  and the deterministic Boolean function  $ITE$  taking as input a bit  $b$ , two valuations  $val$  and  $val'$ , a program  $\alpha$  and a sequence  $d$  of  $k$  bits. Let  $b$  be a bit,  $val$  and  $val'$  be two valuations and  $\alpha$  be a program. The intuitive meaning of these functions will be explained later. The deterministic Boolean function  $REL$  is defined as follows:

```
function  $REL(b, val, val', \alpha)$  returns Boolean
begin
case  $(b, \alpha)$  of
   $(0, +p)$ :  $bool := "val' \neq val \cup \{p\}"$ ;
   $(1, +p)$ :  $bool := "val' = val \cup \{p\}"$ ;
   $(0, -p)$ :  $bool := "val' \neq val \setminus \{p\}"$ ;
   $(1, -p)$ :  $bool := "val' = val \setminus \{p\}"$ ;
   $(0, \beta; \gamma)$ :
    begin
       $bool := true$ ;
       $val'' := 0 \dots 0$ ;
      repeat until  $bool = false$  or  $val'' = 0 \dots 0$ 
      begin
         $bool := REL(0, val, val'', \beta)$  or  $REL(0, val'', val', \gamma)$ ;
```



```

         $val'' := val'' + 1$ 
    end;
end;
(1,  $\beta; \gamma$ ):
begin
     $bool := false$ ;
     $val'' := 0 \dots 0$ ;
    repeat until  $bool = true$  or  $val'' = 0 \dots 0$ 
    begin
         $bool := REL(1, val, val'', \beta)$  and  $REL(1, val'', val', \gamma)$ ;
         $val'' := val'' + 1$ 
    end;
end;
(0,  $\beta \cup \gamma$ ):
     $bool := REL(0, val, val', \beta)$  and  $REL(0, val, val', \gamma)$ ;
(1,  $\beta \cup \gamma$ ):
     $bool := REL(1, val, val', \beta)$  or  $REL(1, val, val', \gamma)$ ;
(0,  $\beta^*$ ):
begin
     $bool := true$ ;
     $d := 0 \dots 0$ ;
    repeat until  $bool = false$  or  $d = 0 \dots 0$ 
    begin
         $bool := ITE(0, val, val', \beta, d)$ ;
         $d := d + 1$ 
    end;
end;
(1,  $\beta^*$ ):
begin
     $bool := false$ ;
     $d := 0 \dots 0$ ;
    repeat until  $bool = true$  or  $d = 0 \dots 0$ 
    begin
         $bool := ITE(1, val, val', \beta, d)$ ;
         $d := d + 1$ 
    end;
end;
(0,  $\phi?$ ):
     $bool := "val \neq val'"$  or  $MOD(0, val', \phi)$ ;
(1,  $\phi?$ ):
     $bool := "val = val'"$  and  $MOD(1, val', \phi)$ 
end case;
return  $bool$ 
end;

```

Let  $b$  be a bit,  $val$  be a formula and  $\varphi$  be a formula. The deterministic

Boolean function  $MOD$  is defined as follows:

```

function  $MOD(b, val, \varphi)$  returns Boolean
begin
case  $(b, \varphi)$  of
   $(0, p)$ :  $bool := "p \notin val"$ ;
   $(1, p)$ :  $bool := "p \in val"$ ;
   $(0, \perp)$ :  $bool := true$ ;
   $(1, \perp)$ :  $bool := false$ ;
   $(0, [\alpha]\phi)$ :
    begin
       $bool := false$ ;
       $val' := 0 \dots 0$ ;
      repeat until  $bool = true$  or  $val' = 0 \dots 0$ 
      begin
         $bool := REL(1, val, val', \alpha)$  and  $MOD(0, val', \phi)$ ;
         $val' := val' + 1$ 
      end;
    end;
   $(1, [\alpha]\phi)$ :
    begin
       $bool := true$ ;
       $val' := 0 \dots 0$ ;
      repeat until  $bool = false$  or  $val' = 0 \dots 0$ 
      begin
         $bool := REL(0, val, val', \alpha)$  or  $MOD(1, val', \phi)$ ;
         $val' := val' + 1$ 
      end;
    end;
end case;
return  $bool$ 
end;

```

Let  $b$  be a bit,  $val$  and  $val'$  be two valuations,  $\alpha$  be a program and  $d$  a sequence of  $k$  bits. The deterministic Boolean function  $ITE$  is defined as follows:

```

function  $ITE(b, val, val', \alpha, d)$  returns Boolean
begin
case  $(b, d)$  of
   $(0, 0 \dots 0)$ :  $bool := "val \neq val'"$ ;
   $(1, 0 \dots 0)$ :  $bool := "val = val'"$ ;
   $(0, \text{odd integer})$ :
    begin
       $bool := true$ ;
       $val'' := 0 \dots 0$ ;
      repeat until  $bool = false$  or  $val'' = 0 \dots 0$ 
      begin

```

```

     $bool := REL(0, val, val'', \alpha)$  or  $ITE(0, val'', val', \alpha, d - 1)$ ;
     $val'' := val'' + 1$ 
end;
end;
(1, odd integer):
begin
     $bool := false$ ;
     $val'' := 0 \dots 0$ ;
    repeat until  $bool = true$  or  $val'' = 0 \dots 0$ 
    begin
         $bool := REL(1, val, val'', \alpha)$  and  $ITE(1, val'', val', \alpha, d - 1)$ ;
         $val'' := val'' + 1$ 
    end;
end;
(0, even integer):
begin
     $bool := true$ ;
     $val'' := 0 \dots 0$ ;
    repeat until  $bool = false$  or  $val'' = 0 \dots 0$ 
    begin
         $bool := ITE(0, val, val'', \alpha, d/2)$  or  $ITE(0, val'', val', \alpha, d/2)$ ;
         $val'' := val'' + 1$ 
    end;
end;
(1, even integer):
begin
     $bool := false$ ;
     $val'' := 0 \dots 0$ ;
    repeat until  $bool = true$  or  $val'' = 0 \dots 0$ 
    begin
         $bool := ITE(1, val, val'', \alpha, d/2)$  and  $ITE(1, val'', val', \alpha, d/2)$ ;
         $val'' := val'' + 1$ 
    end;
end;
end case;
return  $bool$ 
end;

```

The deterministic Boolean function  $REL$  takes as input a bit  $b$ , two valuations  $val$  and  $val'$  and a program  $\alpha$ . Its termination guarantees the following:

- if  $REL(b, val, val', \alpha)$  returns “true”, then either  $b = 0$  and  $(val, val') \notin \|\alpha\|$ , or  $b = 1$  and  $(val, val') \in \|\alpha\|$ ,
- if  $REL(b, val, val', \alpha)$  returns “false”, then either  $b = 0$  and  $(val, val') \in \|\alpha\|$ , or  $b = 1$  and  $(val, val') \notin \|\alpha\|$ .

The deterministic Boolean function  $MOD$  takes as input a bit  $b$ , a valuation  $val$  and a formula  $\varphi$ . Its termination should guarantee the following:

- if  $MOD(b, val, \alpha)$  returns “true”, then either  $b = 0$  and  $val \notin \|\varphi\|$ , or  $b = 1$  and  $val \in \|\varphi\|$ ,
- if  $MOD(b, val, \alpha)$  returns “false”, then either  $b = 0$  and  $val \in \|\varphi\|$ , or  $b = 1$  and  $val \notin \|\varphi\|$ .

The deterministic Boolean function  $ITE$  takes as input a bit  $b$ , two valuations  $val$  and  $val'$ , a program  $\alpha$  and a sequence  $d$  of  $k$  bits. We identify the sequence  $d$  and the integer represented by  $d$ . Its termination should guarantee the following:

- if  $ITE(b, val, val', \alpha, d)$  returns “true”, then either  $b = 0$  and  $(val, val') \notin \|\alpha^d\|$ , or  $b = 1$  and  $(val, val') \in \|\alpha^d\|$ ,
- if  $ITE(b, val, val', \alpha, d)$  returns “false”, then either  $b = 0$  and  $(val, val') \in \|\alpha^d\|$ , or  $b = 1$  and  $(val, val') \notin \|\alpha^d\|$ .

### 5.3 Soundness and completeness

Let  $\Gamma = \text{Pgm}(PV) \times \text{Fml}(PV) \times \overline{K}$  where  $\overline{K}$  is the set of all sequences of  $k$  bits. We define the binary relation  $\ll$  on  $\Gamma$  in the following way:  $(\alpha, \phi, d) \ll (\beta, \psi, e)$  iff one of following condition holds:

- $len(\alpha) + len(\phi) < len(\beta) + len(\psi)$ ,
- $len(\alpha) + len(\phi) = len(\beta) + len(\psi)$  and  $d < e$ .

**Lemma 1.**  $\ll$  is a well-founded strict partial order on  $\Gamma$ .

*Proof.* By the well-foundedness of the standard linear order between non-negative integers.  $\square$

Let  $\Sigma$  be the set of all  $(\alpha, \phi, d) \in \Gamma$  such that the following condition holds:

1. for all bits  $b$  and for all valuations  $val$  and  $val'$ ,
  - if  $REL(b, val, val', \alpha)$  returns “true”, then either  $b = 0$  and  $(val, val') \notin \|\alpha\|$ , or  $b = 1$  and  $(val, val') \in \|\alpha\|$ ,
  - if  $REL(b, val, val', \alpha)$  returns “false”, then either  $b = 0$  and  $(val, val') \in \|\alpha\|$ , or  $b = 1$  and  $(val, val') \notin \|\alpha\|$ ,
2. for all bits  $b$  and for all valuations  $val$ ,
  - if  $MOD(b, val, \phi)$  returns “true”, then either  $b = 0$  and  $val \notin \|\phi\|$ , or  $b = 1$  and  $val \in \|\phi\|$ ,
  - if  $MOD(b, val, \phi)$  returns “false”, then either  $b = 0$  and  $val \in \|\phi\|$ , or  $b = 1$  and  $val \notin \|\phi\|$ ,

3. for all bits  $b$  and for all valuations  $val$  and  $val'$ ,
- if  $ITE(b, val, val', \alpha, d)$  returns “true”, then either  $b = 0$  and  $(val, val') \notin \|\alpha^d\|$ , or  $b = 1$  and  $(val, val') \in \|\alpha^d\|$ ,
  - if  $ITE(b, val, val', \alpha, d)$  returns “false”, then either  $b = 0$  and  $(val, val') \in \|\alpha^d\|$ , or  $b = 1$  and  $(val, val') \notin \|\alpha^d\|$ .

The aim is to prove by  $\ll$ -induction that all  $(\alpha, \phi, d)$  are in  $\Sigma$ . As lemma 1 states that  $\ll$  is a well-founded strict partial order, it is sufficient to prove the following lemma.

**Lemma 2.** *Let  $(\alpha, \phi, d) \in \Gamma$ . If*

$$\text{for all } (\beta, \psi, e) \in \Gamma, \text{ if } (\beta, \psi, e) \ll (\alpha, \phi, d), \text{ then } (\beta, \psi, e) \in \Sigma \quad (\text{H})$$

*then  $(\alpha, \phi, d) \in \Sigma$ .*

*Proof.* Suppose (H).

(1.) **The function REL.** Let  $b$  be a bit and  $val$  and  $val'$  be valuations. Suppose  $REL(b, val, val', \alpha)$  returns “true”. We have to consider different cases.

**Cases**  $(b, \alpha) = (0, +p)$ , **or**  $(b, \alpha) = (1, +p)$ , **or**  $(b, \alpha) = (0, -p)$  **and**  $(b, \alpha) = (1, -p)$ . Left to the reader.

**Case**  $(b, \alpha) = (0, \beta; \gamma)$ . Hence,  $b = 0$  and  $\alpha = \beta; \gamma$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then for all valuations  $val''$ , either  $REL(0, val, val'', \beta)$  returns “true”, or  $REL(0, val'', val', \gamma)$  returns “true”. Remark that  $(\beta, \phi, d) \ll (\alpha, \phi, d)$  and  $(\gamma, \phi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \phi, d) \in \Sigma$  and  $(\gamma, \phi, d) \in \Sigma$ . Since for all valuations  $val''$ , either  $REL(0, val, val'', \beta)$  returns “true”, or  $REL(0, val'', val', \gamma)$  returns “true”, then for all valuations  $val''$ , either  $(val, val'') \notin \|\beta\|$ , or  $(val'', val') \notin \|\gamma\|$ . Thus,  $(val, val') \notin \|\alpha\|$ .

**Case**  $(b, \alpha) = (1, \beta; \gamma)$ . Hence,  $b = 1$  and  $\alpha = \beta; \gamma$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then there exists a valuation  $val''$  such that  $REL(b, val, val'', \beta)$  returns “true” and  $REL(b, val'', val', \gamma)$  returns “true”. Remark that  $(\beta, \phi, d) \ll (\alpha, \phi, d)$  and  $(\gamma, \phi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \phi, d) \in \Sigma$  and  $(\gamma, \phi, d) \in \Sigma$ . Since there exists a valuation  $val''$  such that  $REL(1, val, val'', \beta)$  returns “true” and  $REL(1, val'', val', \gamma)$  returns “true”, then there exists a valuation  $val''$  such that  $(val, val'') \in \|\beta\|$  and  $(val'', val') \in \|\gamma\|$ . Thus,  $(val, val') \in \|\alpha\|$ .

**Cases**  $(b, \alpha) = (0, \beta \cup \gamma)$  **and**  $(b, \alpha) = (1, \beta \cup \gamma)$ . These cases are similarly treated.

**Case**  $(b, \alpha) = (0, \beta^*)$ . Hence,  $b = 0$  and  $\alpha = \beta^*$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then for all sequences  $e$  of  $k$  bits,  $ITE(0, val, val', \beta, e)$  returns “true”. Remark that  $(\beta, \phi, e) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \phi, e) \in \Sigma$ . Since for all sequences  $e$  of  $k$  bits,  $ITE(0, val, val', \beta, e)$  returns “true”, then for all sequences  $e$  of  $k$  bits,  $(val, val') \notin \|\beta^e\|$ . Thus,  $(val, val') \notin \|\alpha\|$ .

**Case**  $(b, \alpha) = (1, \beta^*)$ . Hence,  $b = 1$  and  $\alpha = \beta^*$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then there exists a sequence  $e$  of  $k$  bits such that  $ITE(1, val, val', \beta, e)$  returns “true”. Remark that  $(\beta, \phi, e) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \phi, e) \in \Sigma$ . Since there exists a sequence  $e$  of  $k$  bits such that

$ITE(1, val, val', \beta, e)$  returns “true”, then there exists a sequence  $e$  of  $k$  bits such that  $(val, val') \in \|\beta^e\|$ . Thus,  $(val, val') \in \|\alpha\|$ .

**Case**  $(b, \alpha) = (0, \psi?)$ . Hence,  $b = 0$  and  $\alpha = \psi?$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then either  $val \neq val'$ , or  $MOD(0, val', \psi)$  returns “true”. In the former case,  $(val, val') \notin \|\alpha\|$ . In the latter case, remark that  $(+p, \psi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(+p, \psi, d) \in \Sigma$ . Since  $MOD(0, val', \psi)$  returns “true”, then  $val' \notin \|\psi\|$ . Thus,  $(val, val') \notin \|\alpha\|$ .

**Case**  $(b, \alpha) = (1, \psi?)$ . Hence,  $b = 1$  and  $\alpha = \psi?$ . Since  $REL(b, val, val', \alpha)$  returns “true”, then  $val = val'$  and  $MOD(1, val', \psi)$  returns “true”. Remark that  $(+p, \psi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(+p, \psi, d) \in \Sigma$ . Since  $MOD(1, val', \psi)$  returns “true”, then  $val' \in \|\psi\|$ . Since  $val = val'$ , then  $(val, val') \in \|\alpha\|$ .

Suppose  $REL(b, val, val', \alpha)$  returns “false”. We have to consider cases similar to the above ones.

(2.) **The function MOD.** Let  $b$  be a bit and  $val$  be a valuation.

Suppose  $MOD(b, val, \phi)$  returns “true”. We have to consider several cases.

**Cases**  $(b, \phi) = (0, p)$ , **or**  $(b, \phi) = (1, p)$ , **or**  $(b, \phi) = (0, \perp)$  **and**  $(b, \phi) = (1, \perp)$ . Left to the reader.

**Case**  $(b, \phi) = (0, [\beta]\psi)$ . Hence,  $b = 0$  and  $\phi = [\beta]\psi$ . Since  $MOD(b, val, \phi)$  returns “true”, then there exists a valuation  $val'$  such that  $REL(1, val, val', \beta)$  returns “true” and  $MOD(0, val', \psi)$  returns “true”. Remark that  $(\beta, \psi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \psi, d) \in \Sigma$ . Since there exists a valuation  $val'$  such that  $REL(1, val, val', \beta)$  returns “true” and  $MOD(0, val', \psi)$  returns “true”, then there exists a valuation  $val'$  such that  $(val, val') \in \|\beta\|$  and  $val' \notin \|\psi\|$ . Thus,  $val \notin \|\phi\|$ .

**Case**  $(b, \phi) = (1, [\beta]\psi)$ . Hence,  $b = 1$  and  $\phi = [\beta]\psi$ . Since  $MOD(b, val, \phi)$  returns “true”, then for all valuations  $val'$ , either  $REL(0, val, val', \beta)$  returns “true”, or  $MOD(1, val', \psi)$  returns “true”. Remark that  $(\beta, \psi, d) \ll (\alpha, \phi, d)$ . Since (H), then  $(\beta, \psi, d) \in \Sigma$ . Since for all valuations  $val'$ , either  $REL(0, val, val', \beta)$  returns “true”, or  $MOD(1, val', \psi)$  returns “true”, then for all valuations  $val'$ , either  $(val, val') \notin \|\beta\|$ , or  $val' \in \|\psi\|$ . Thus,  $val \in \|\phi\|$ .

Suppose  $MOD(b, val, \phi)$  returns “false”. We have to consider cases similar to the above ones.

(3.) **The function ITE.** Let  $b$  be a bit and  $val$  and  $val'$  be valuations.

Suppose  $ITE(b, val, val', \alpha, d)$  returns “true”. We have to consider several cases.

**Cases**  $(b, d) = (0, 0 \dots 0)$ , **or**  $(b, d) = (1, 0 \dots 0)$ . Left to the reader.

**Case**  $(b, d) = (0, \text{odd integer})$ . Hence,  $b = 0$  and  $d = e1$  for some sequence  $e$  of  $k - 1$  bits. Since  $ITE(b, val, val', \alpha, d)$  returns “true”, then for all valuations  $val''$ , either  $REL(0, val, val'', \alpha)$  returns “true”, or  $ITE(0, val'', val', \alpha, e)$  returns “true”. Remark that  $(\alpha, \phi, e) \ll (\alpha, \phi, d)$ . Since (H), then  $(\alpha, \phi, e) \in \Sigma$ . Since for all valuations  $val''$ , either  $REL(0, val, val'', \alpha)$  returns “true”, or  $ITE(0, val'', val', \alpha, e)$  returns “true”, then for all valuations  $val''$ , either  $(val, val'') \notin \|\alpha\|$ , or  $(val'', val') \notin \|\alpha^e\|$ . Thus,  $(val, val') \notin \|\alpha^d\|$ .

**Case**  $(b, d) = (1, \text{odd integer})$ . Hence,  $b = 1$  and  $d = e1$  for some sequence  $e$  of  $k - 1$  bits. Since  $ITE(b, val, val', \alpha, d)$  returns “true”, then there exists a valuation  $val''$  such that  $REL(1, val, val'', \alpha)$  returns “true” and

$ITE(1, val'', val', \alpha, e0)$  returns “true”. Remark that  $(\alpha, \phi, e0) \ll (\alpha, \phi, d)$ . Since (H), then  $(\alpha, \phi, e0) \in \Sigma$ . Since there exists a valuation  $val''$  such that  $REL(1, val, val'', \alpha)$  returns “true” and  $ITE(1, val'', val', \alpha, e0)$  returns “true”, then there exists a valuation  $val''$  such that  $(val, val'') \in \|\alpha\|$  and  $(val'', val') \in \|\alpha^{e0}\|$ . Thus,  $(val, val') \in \|\alpha^d\|$ .

**Case  $(b, d) = (0, \text{even integer})$ .** Hence,  $b = 0$  and  $d = e0$  for some sequence  $e$  of  $k - 1$  bits. Since  $ITE(b, val, val', \alpha, d)$  returns “true”, then for all valuations  $val''$ , either  $ITE(0, val, val'', \alpha, 0e)$  returns “true”, or  $ITE(0, val'', val', \alpha, 0e)$  returns “true”. Remark that  $(\alpha, \phi, 0e) \ll (\alpha, \phi, d)$ . Since (H), then  $(\alpha, \phi, 0e) \in \Sigma$ . Since for all valuations  $val''$ , either  $ITE(0, val, val'', \alpha, 0e)$  returns “true”, or  $ITE(0, val'', val', \alpha, 0e)$  returns “true”, then for all valuations  $val''$ , either  $(val, val'') \in \|\alpha^{0e}\|$ , or  $(val'', val') \in \|\alpha^{0e}\|$ . Thus,  $(val, val') \in \|\alpha^d\|$ .

**Case  $(b, d) = (1, \text{even integer})$ .** Hence,  $b = 1$  and  $d = e0$  for some sequence  $e$  of  $k - 1$  bits. Since  $ITE(b, val, val', \alpha, d)$  returns “true”, then there exists a valuation  $val''$  such that  $ITE(1, val, val'', \alpha, 0e)$  returns “true” and  $ITE(1, val'', val', \alpha, 0e)$  returns “true”. Remark that  $(\alpha, \phi, 0e) \ll (\alpha, \phi, d)$ . Since (H), then  $(\alpha, \phi, 0e) \in \Sigma$ . Since there exists a valuation  $val''$  such that  $ITE(1, val, val'', \alpha, 0e)$  returns “true” and  $ITE(1, val'', val', \alpha, 0e)$  returns “true”, then there exists a valuation  $val''$  such that  $(val, val'') \in \|\alpha^{0e}\|$  and  $(val'', val') \in \|\alpha^{0e}\|$ . Thus,  $(val, val') \in \|\alpha^d\|$ .

Suppose  $ITE(b, val, val', \alpha, d)$  returns “false”. We have to consider cases similar to the above ones.  $\square$

**Proposition 2.**  $\Sigma = \Gamma$ .

*Proof.* By Lemmas 1 and 2.  $\square$

Hence, the functions  $REL$ ,  $MOD$  and  $ITE$  are sound and complete.

## 5.4 Complexity

For all programs  $\alpha$ , let  $f_{REL}(\alpha)$  be the maximal number of recursive calls between  $REL$ ,  $MOD$  and  $ITE$  within the context of a call of the form  $REL(b, val, val', \alpha)$ . For all formulas  $\varphi$ , let  $f_{MOD}(\varphi)$  be the maximal number of recursive calls between  $REL$ ,  $MOD$  and  $ITE$  within the context of a call of the form  $MOD(b, val, \varphi)$ . For all programs  $\alpha$ , let  $f_{ITE}(\alpha)$  be the maximal number of recursive calls between  $REL$ ,  $MOD$  and  $ITE$  within the context of a call of the form  $ITE(b, val, val', \alpha, d)$ .

**Claim 1.**  $f_{ITE}(\alpha) \leq f_{REL}(\alpha) + 2 \times k - 1$ .

*Proof.* Obvious.  $\square$

**Claim 2.**  $f_{REL}(\alpha) \leq 2 \times \text{len}(\alpha) \times k$  and  $f_{MOD}(\varphi) \leq 2 \times \text{len}(\varphi) \times k$ .

*Proof.* Let  $\Pi$  be the property that holds for a pair  $(\alpha, \varphi)$  iff  $f_{REL}(\alpha) \leq 2 \times \text{len}(\alpha) \times k$  and  $f_{MOD}(\varphi) \leq 2 \times \text{len}(\varphi) \times k$ . Let  $\ll$  be the binary relation that holds between pairs  $(\alpha, \varphi)$  and  $(\alpha', \varphi')$  iff either  $\text{len}(\alpha) < \text{len}(\alpha')$  and  $\text{len}(\varphi)$

$\leq \text{len}(\varphi')$ , or  $\text{len}(\alpha) \leq \text{len}(\alpha')$  and  $\text{len}(\varphi) < \text{len}(\varphi')$ . Remark that  $\ll$  is a well-founded order. Let us demonstrate by  $\ll$ -induction that  $\Pi$  holds for all pairs  $(\alpha, \varphi)$ . Let  $(\alpha, \varphi)$  be such that for all  $(\alpha', \varphi')$ , if  $(\alpha', \varphi') \ll (\alpha, \varphi)$ , then  $\Pi$  holds for  $(\alpha', \varphi')$ . We only consider the following 2 cases.

**Case  $\alpha = \beta^*$ .** Obviously,  $f_{REL}(\beta^*) = f_{ITE}(\beta) + 1$ . By Claim 1,  $f_{ITE}(\beta) \leq f_{REL}(\beta) + 2 \times k - 1$ . By induction hypothesis,  $f_{REL}(\beta) \leq 2 \times \text{len}(\beta) \times k$ . Hence,  $f_{REL}(\beta^*) \leq 2 \times (\text{len}(\beta) + 1) \times k \leq 2 \times \text{len}(\beta^*) \times k$ .

**Case  $\varphi = [\beta]\phi$ .** Obviously,  $f_{MOD}([\beta]\phi) \leq \max\{f_{REL}(\beta), f_{MOD}(\phi)\} + 1$ . By induction hypothesis,  $f_{REL}(\beta) \leq 2 \times \text{len}(\beta) \times k$  and  $f_{MOD}(\phi) \leq 2 \times \text{len}(\phi) \times k$ . Hence,  $f_{MOD}([\beta]\phi) \leq 2 \times \max\{\text{len}(\beta), \text{len}(\phi)\} \times k + 1 \leq 2 \times \text{len}([\beta]\phi) \times k$ .  $\square$

Hence the maximal number of recursive calls between the deterministic Boolean functions  $MOD$ ,  $REL$  and  $ITE$  has order linear in  $k + \text{len}(\varphi) + \text{len}(\alpha)$ . Thus they can be implemented on deterministic Turing machines running in polynomial space.

This concludes the proof that our model checking algorithm works in polynomial space.

## 6 Conclusion

We have clarified the complexity of the model checking and the satisfiability problem of Dynamic Logic of Propositional Assignments (DL-PA) and of Coalition Logic of Propositional Control and Delegation DCL-PC. First, we have explained why the proof of EXPTIME-hardness of the DL-PA model checking problem presented in [1, Thm 4] is erroneous. Second, although DCL-PC model checking is indeed in PSPACE, its proof in [9, Thm. 4] is flawed, and we have given a correct proof that the model checking and the satisfiability problem of both DL-PA and DCL-PC are in PSPACE. All upper bounds are tight because the problem QSAT can be translated into the DL-PA model checking problem, as shown in [4].

## References

- [1] Philippe Balbiani, Andreas Herzig, and Nicolas Troquard. Dynamic logic of propositional assignments: A well-behaved variant of pdl. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 143–152. IEEE Computer Society, 2013.
- [2] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981.
- [3] Olivier Gasquet, Valentin Goranko, and François Schwarzentruber. Big brother logic: logical modeling and reasoning about agents equipped with surveillance cameras in the plane. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 325–332, 2014.



- [4] Andreas Herzig, Emiliano Lorini, Frédéric Moisan, and Nicolas Troquard. A dynamic logic of normative systems. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, pages 228–233. IJCAI/AAAI, 2011.
- [5] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.
- [6] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.
- [7] Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.
- [8] Michael L. Tiomkin and Johann A. Makowsky. Propositional dynamic logic with local assignments. *Theor. Comput. Sci.*, 36:71–87, 1985.
- [9] Wiebe van der Hoek, Dirk Walther, and Michael Wooldridge. Reasoning about the transfer of control. *J. Artif. Intell. Res. (JAIR)*, 37:437–477, 2010.

## Appendix

--- In (Balbiani, Herzig, Troquard, 2013 LICS) a supposedly polynomial  
--- reduction from the problem PEEK-G5 (Stockemeyer Chandra 1979) into  
--- the model checking problem in the logic of DL-PA is proposed. If  
--- the reduction were actually working, a similar reduction could be  
--- done from PEEK-G5 into the model checking problem of CTL over  
--- NuSMV models.

--- We consider here the Peek instance where Eloise controls ep1, and  
--- Abelard controls ap1 and ap2. Abelard plays first (Tau = A). The  
--- goal formula Phi for this instance is ep1 & ap1. The valuation to  
--- start with is empty: ep1, ap1, and ap2 are set to false. Clearly,  
--- if Abelard never assigns true to ap1, Phi can never become  
--- true. So clearly, Eloise does not have a winning strategy. So,  
--- were the reduction working, we should not find a counter-model  
--- when evaluating the present file. But a counter-model is found. So  
--- the reduction in (Balbiani, Herzig, Troquard, 2013 LICS) is wrong.

MODULE abelard(turn, Phi)

--- Abelard controls two variables ap1 and ap2, both initially set to  
--- false. Abelard can non-deterministically choose which variable to  
--- change before his turn, that is, when it is the turn of  
--- eloise. This is done by setting vartochange-a to either 1 or  
--- 2. Then Abelard can set ap1 (next(ap1)) to either true or false,  
--- when vartochange = 1, it is his turn (turn = a), and Phi is not  
--- true. Abelard can set ap2 (next(ap2)) to either true or false, when  
--- vartochange = 2, it is his turn (turn = a), and Phi is not true.

VAR

vartochange-a : {1,2};  
ap1 : boolean;  
ap2 : boolean;

ASSIGN

init(vartochange-a) := {1,2};  
init(ap1) := FALSE;  
init(ap2) := FALSE;  
next(vartochange-a) := (!Phi & turn = e) ? {1,2}: vartochange-a;  
next(ap1) := (!Phi & turn = a & vartochange-a = 1) ? {TRUE, FALSE} : ap1;  
next(ap2) := (!Phi & turn = a & vartochange-a = 2) ? {TRUE, FALSE} : ap2;

```

MODULE eloise(turn, Phi)

--- Eloise controls only one variable ep1. Its initial value is set to
--- false. Eloise can set the value of ep1 (next(ep1)) to either true
--- or false, whenever it is her turn (turn = e) and Phi is not
--- true. Since she controls only one variable, the control variable
--- vartochange-e is dummy, but is used for uniformity with the MODULE
--- abelard.

VAR
  vartochange-e : {1};
  ep1 : boolean;
ASSIGN
  init(ep1) := FALSE;
  next(ep1) := (!Phi & turn = e & vartochange-e = 1) ? {TRUE, FALSE} : ep1;

MODULE main
  VAR
    turn : {e,a};
    nowin : boolean;

--- We consider here a Peek instance where Eloise controls ep1, and
--- Abelard controls ap1 and ap2. The valuation to start with is
--- empty: ep1, ap1, and ap2 are all set to false. In other words, elo
--- is an instance of the module eloise, and abe is an instance of the
--- module abelard; both defined in this file.

    elo : eloise(turn, Phi);
    abe : abelard(turn, Phi);

  DEFINE

--- In the Peek instance we consider, Abelard plays first (Tau =
--- A). The objective formula Phi is ep1 & ap1.

  Phi := (elo.ep1) & (abe.ap2);
  Tau := a;

  ASSIGN

    init(turn) := Tau;
    init(nowin) := TRUE;

    next(turn) :=

```

```

case
    (turn = e) : a;
    (turn = a) : e;
esac;

next(nowin) := Phi ? FALSE : nowin;

CTLSPEC

--- This formula is an immediate translation of the DL-PA formula in
--- (Balbiani, Herzig, Troquard, 2013 LICS) into the language of CTL.

    AG (nowin -> (
!Phi                                     &
((turn = e) -> AX nowin) &
((turn = a) -> EX nowin)
    )
    )

```